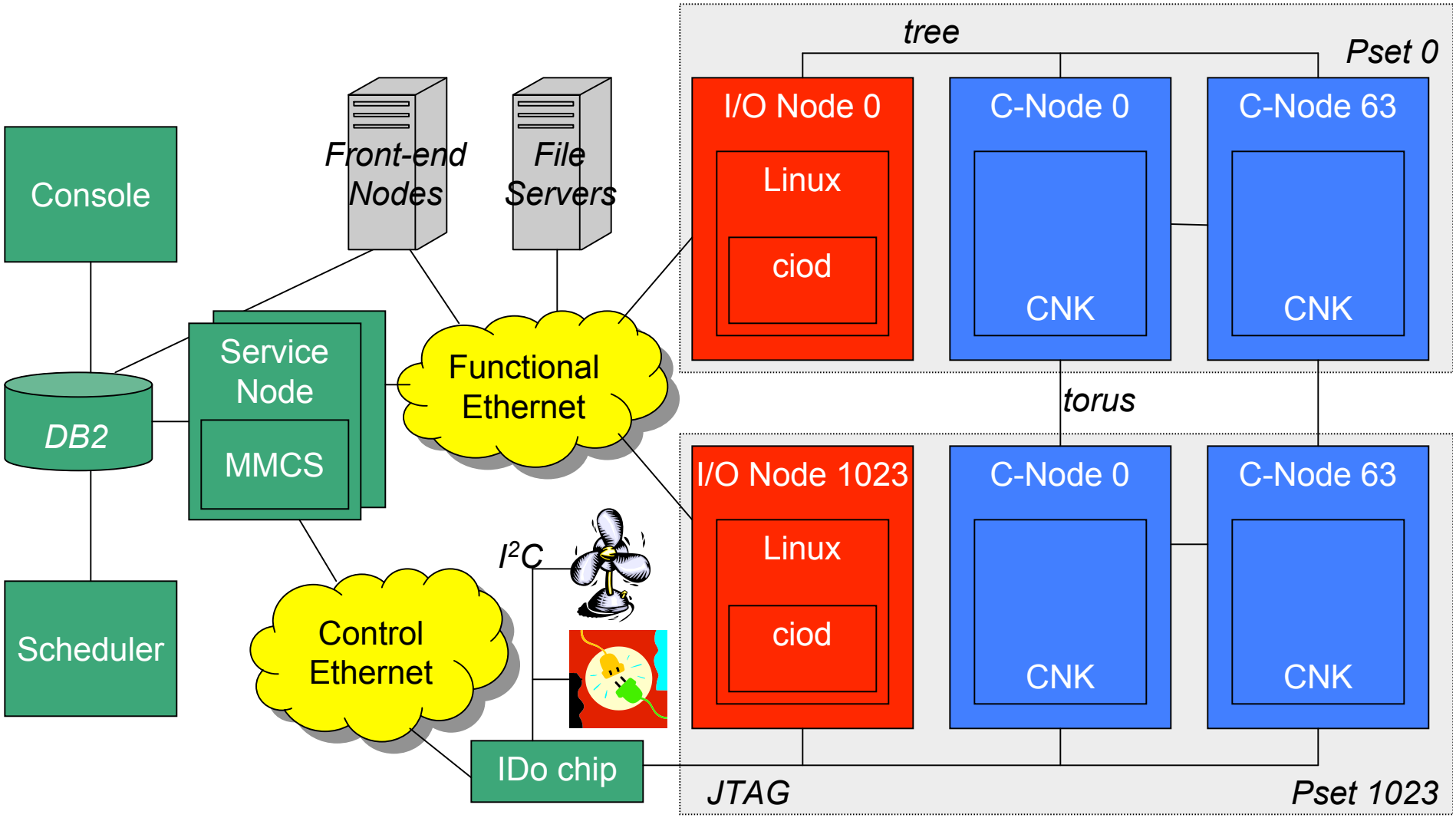
IBM Research

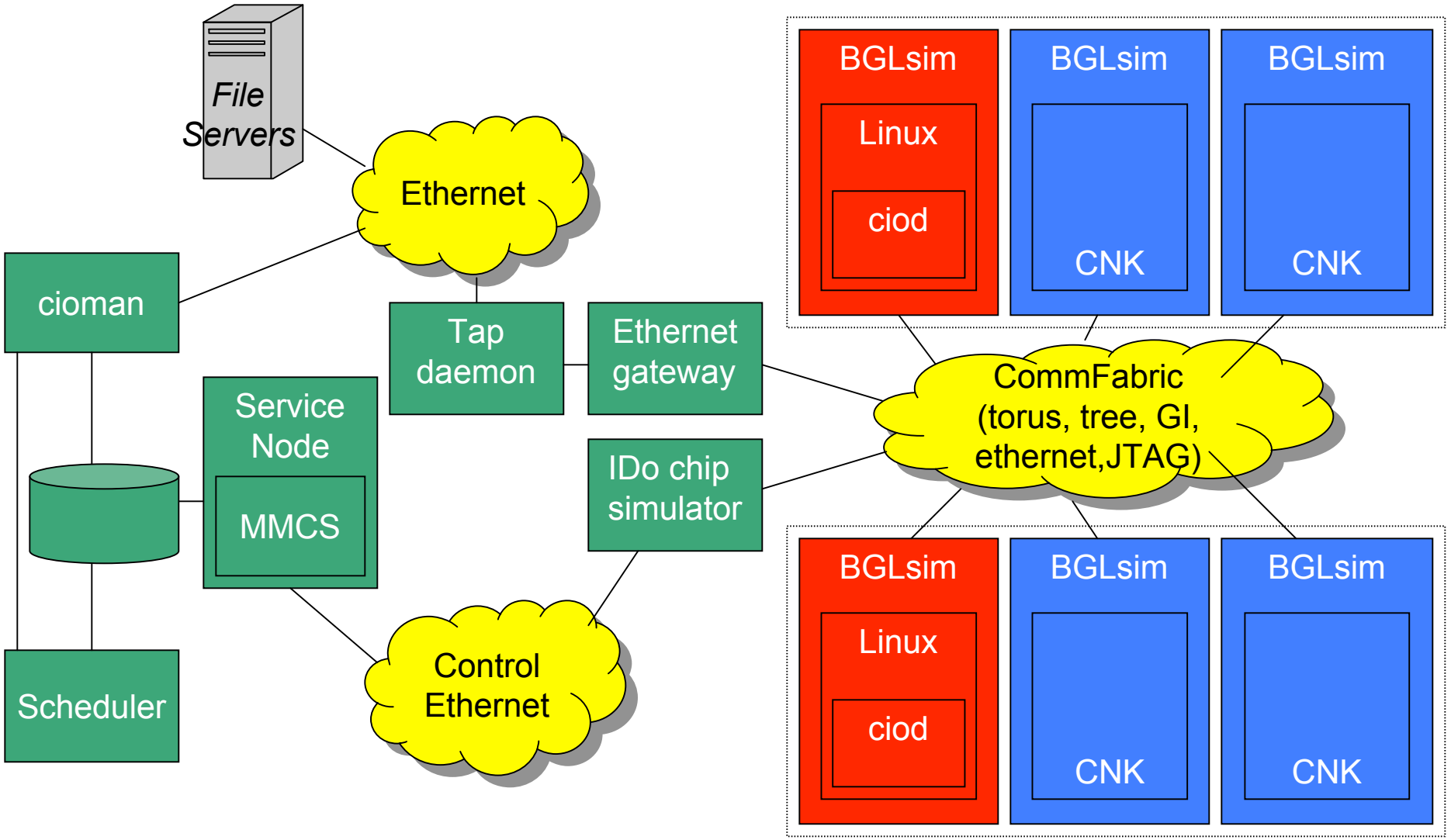
Using the BG/L Simulation Environment

George Almási, Ralph Bellofatto, José Brunheroto,
José Casta_os, Luis Ceze, Paul Crumley,
Derek Lieber, José Moreira, Alda Sanomiya,
Karin Strauss
… and many others

Blue Gene/L System Software Architecture



Blue Gene/L Simulation Environment



BGLSim Overview

- Architectural simulator of a single BG/L node
  - Consumes PPC440 binaries
  - One cycle per instruction
  - Statistics as instruction histograms, traces; timing model
  - Runs on Linux/x86 workstations
- BG/L specific features:
  - Supports 2 PPC 440 cores per chip
  - 440GP instruction set
  - Hummer^2 (Oedipus ISA) floating point
  - Architecture accurate caches
    - L1, L2, L3
  - EMAC4, MAL (1Gb/s Ethernet)
  - BG/L interrupt controller (BIC)
  - Torus, tree devices and other networks

`BGLSim` Invocation in Single Chip Mode

`mambo [options]`

- Verbose mode (`-v`): print every instruction executed
- Verbose interrupts (`-z`): print every interrupt
- Single/dual core mode (`-S,-D`)
- Cache model (`-L:123,12,13,None`)
- PseudoUART console (`-x`): interactive console under Linux
- Interactive mode (`-i`): CTRL-C suspends the simulator
  - Peek, poke memory, registers, TLBs
- Preload ELF images (`-e`)
  - Significantly faster than loading them through JTAG
- Torus/tree cheat (`-t`)
  - Preconfigure torus and tree

PreLoading ELF Images into `BGLSim`

- **mambo -e file1,file2,file3…**
  - BGLSim preloads sequences of ELF images into memory
  - Do not put spaces between files names!

- Booting linux
  - -e `sram.bin,Image.initrd.elf`
- Booting blrts
  - -e `rtsbooter.rts,rts.rts`
- Booting w/o an operating system
  - Bootstrapper + app in a single file

Simulator Props: Magic Addresses

- A method for invoking simulator functionality directly
- Compatible with Awan/MTI/Cyclesim simulators
  - Same code runs on MTI, Awan and BGLSim
  - Develop code on BGLSim; test on Awan
- Magic **`putchar`**: address 0xEF600300
- Magic **`stop`**: address 0xEF600F00
- s_printf() is built on top of s_putchar()

```
void sim_putchar (char c)
{
  static char *x = 0xEF60300;
  *x = c;
}
```

```
void sim_stop (unsigned exitcode)
{
  static unsigned *x = 0xEF60F00;
  *x = exitcode;
}
```

Multichip simulation architecture: `BGLMachine`

- Machine description file:
  - Racks, midplanes, node cards, compute & I/O cards, ***wiring***
  - Described in XML format
- Used by both the real control system and the simulator
- In real hardware:
  - Backed by a database description of same items
  - Generated from the database
  - Used to create “personality” stamps for individual nodes, tree class routes, torus coordinate assignments
- In simulator:
  - Generated when simulation starts
  - Library accessible to simulation components, esp. **`CommFabric`** and **`simboot`**

Multichip simulation architecture: `simboot`

- "Creates" and IPLs a simulated system
  - Creates a **`BGLMachine`** (machine description file) according to arguments
  - Saves **`BGLMachine`** to a file (**`bglsim.xml`**)
  - Creates and saves an MPI (LAM) schema (**`simboot.schema`**)
    - What programs to run where
    - LLNL ported simboot to use Quadrics (and possibly other) MPI libraries
  - Starts the simulator processes
  - IPLs (boots) the simulators
    - "cheating" – (simulators wake up with pre-loaded images)
    - Alternative boot: simulated control system
- Allows the creation of simulated architectures that have no correspondence in real hardware
  - e.g. 4 compute nodes in a 2x2 torus with 2 I/O nodes
  - A number of hard-coded configurations, as well as command line arguments to create arbitrary* simulations
  - LLNL have created large configurations we didn't previously have

Multichip simulation architecture: `CommFabric`

- "Implements" **`BGLMachine`** in simulator
- Simulates the cabling and network chips in the real hardware
  - All 5 BG/L networks
- **`CommFabric`** is a library linked by all simulation components
- Underlying implementation:
  - MPI messages
    - Torus: packets routed according to hint bits (deterministic or adaptive)
    - Tree: packets routed according to class routes
    - Ethernet: packets routed through Ethernet gateway
    - GI: state changes routed through nodes
  - Kinds of MPI used: LAM (@Watson), Quadrics (@LLNL)

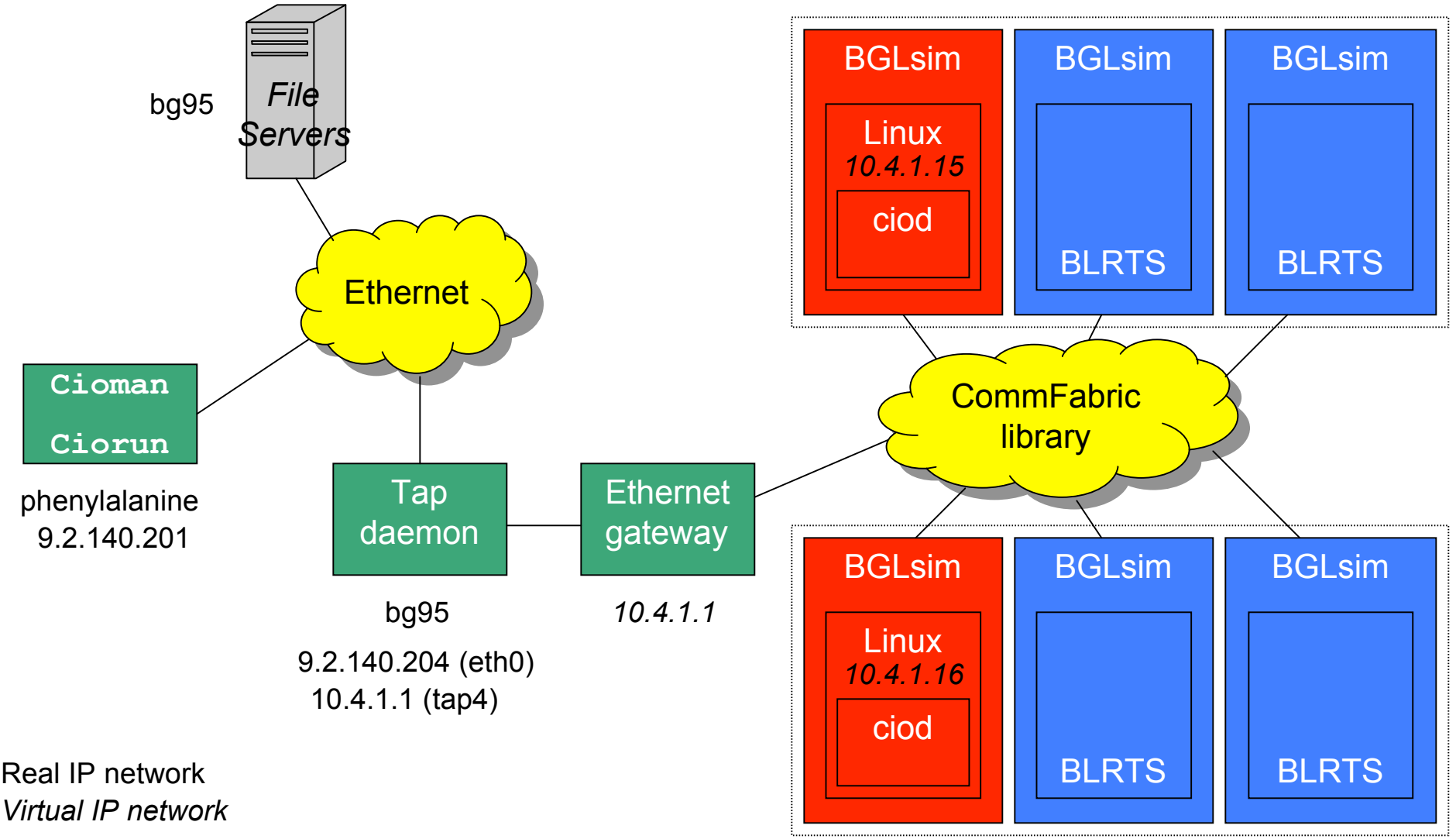
IDo Chip Simulator, MMCS simulator

- IDo sim: Functional simulator of the IDo chip and JTAG network
  - Read, write SRAM
  - Read, write DCRs
  - Apply reset on/off to individual cores
- MMCS_sim: midplane management control system
  - Talks to IDO simulator instead of JTAG network
- MMCS+IDo can boot a simulation
  - simboot starts simulation with all BGLsims running “empty”
  - MMCS loads boot images and applies reset to nodes through IDO sim

TapDaemon and the Ethernet Gateway

- **bglsim** routes external ethernet packets (not 10.0.0.0) to the **TapDaemon** through the `CommFabric` library and the Ethernet gateway
  - Internal Ethernet packets routed directly between mambos through CommFabric
- **TapDaemon**
  - Part of simulation: listens for connections on a well known port
    - Hostname and port number defined when installing simulator
  - Requires root privileges because reads/writes raw ethernet
  - Requires recompilation of the host Linux kernel with TUN/TAP module enabled
  - Log in /var/log/tapserver
- Only one **TapDaemon** shared by all simulations
  - As part of initialization, simboot contacts the tapserver, obtains a new simulation number (called netId) and forks a new tap daemon for the simulation (new functionality)
- Ethernet gateway is the interface between **TapDaemon** and simulation
  - Runs with user privileges
  - Reads and writes `CommFabric` packets (has MPI rank)
  - Reads and writes from/to socket with forked **TapDaemon**

Routing and NFS



cioman and ciorun

- Job starters
- cioman and ciorun run outside the simulation
- Connect to I/O nodes using CIO protocol
  - over real+simulated ethernet
- Once simulation is booted, anybody can connect to it
  - bglsim.xml describes IP addresses of I/O nodes
- `cioman` is interactive, and allows user processes to be debugged
  - “debug” command
- `ciorun` is equivalent to mpirun, and has a –np argument

Installed simulators

- IBM Watson Bluegene simulation cluster
  - ~100 nodes, Intel/Linux, 600MHz-2.4GHz, Cisco switch, LAM MPI
- IBM Rochester simulation cluster
  - ~150 nodes
- bgl.ihost.com
  - 10 nodes, 600MHz Intel/Linux
  - Publicly accessible (outside IBM firewall)
- LLNL ALC cluster

Conclusion

- Simulation environment has a CPU slowdown of ~1000
- Architecturally accurate
- Single-chip simulator, **`bglsim`**, has *timing* model
- Multi-chip simulator deployed in multiple locations
- Proved extremely useful in building system software
  - Setting a new bar in bringing up software along with hardware